UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO DIVISION

ORACLE AMERICA, INC. Plaintiff, v. GOOGLE INC. Defendant.	Case No. CV 10-03561 WHA

SUMMARY AND REPORT OF ROBERT ("BOB") G. VANDETTE

SUBMITTED ON BEHALF OF PLAINTIFF ORACLE AMERICA, INC.

II. BACKGROUND AND QUALIFICATIONS

8. I have over 30 years of experience in designing and developing System Software and Hardware products. During my career, I have been involved in two successful startups and have created my own company which was eventually sold to Sun Microsystems.

9. My focus over most of my career has involved technologies related to emulation and virtualization. I have helped create several successful technologies including VP/ix (Virtual PC on Unix), WABI (Windows ABI Emulator). WABI was the technology that I invented at my own company, Praxsys Technologies, which lead to an acquisition by Sun Microsystems in 1992.

10. Since the Sun Microsystems acquisition, I have been involved in several roles in the development of Sun's Java Virtual Machine technology. I was the lead developer responsible for creating the Java SE Embedded product line at Sun Microsystems. I have focused on Java embedded solutions for the past 6 years. I am currently the Technical Lead Architect for the Java Embedded product line at Oracle Corporation, who acquired Sun Microsystems in 2010.

11. I am qualified to conduct performance benchmark analysis and analyze the results obtained because I have been involved in numerous projects at both Sun Microsystems and Oracle Corporation where I have helped to measure performance, and design and implement improvements in our Java Virtual Machine implementations. Over that last 6 years, as technical lead of the Java SE Embedded products, I have used the exact same benchmarks that are referenced in this report to analyze the performance of our various Virtual Machine implementations against our competitor's products. This analysis typically leads to improvements in areas of the Virtual Machine to ensure that our products stay competitive.

III. INTRODUCTION TO PERFORMANCE BENCHMARKS ANALYSIS AND TESTING ENVIRONMENT

A. Introduction

12. It is common, widely accepted industry practice to use performance benchmark analysis for the purpose of evaluating the relative value of one product over another. Benchmark results are typically posted on a publicly accessible media to allow consumers to make informed decisions on products. Benchmarks are usually designed with a specific focus or workload in mind. They are written to allow quick and repeatable measurements of a function or set of functions that a platform or product might implement. These measurements are then used to compare the speed at which these functions are performed. In the past, one example of where benchmarking was used was to measure the quality of code generated by static language compilers. At the time I used a benchmark known as Dhrystone. The procedure involved compiling and executing this Dhrystone program which then timed itself during the execution. The faster this program ran gave an indication as to how quickly an application using the same compiler would execute. Today I am using dynamic languages, such as Java, which may first interpret and then compile Java byte codes. Since the speed at which a Java program executes is very much dependent on the speed of the interpreter and byte code compiler, benchmarking is used to measure these operations.

13. Customers will select a specific vendor's hardware device, system or platform, that uses a Java virtual machine, based on performance benchmark results. Sun Microsystems (now Oracle America) has focused for years on measuring, improving and publishing benchmarks results for the SPECjbb2000 (retired) and SPECjbb2005 sets of benchmarks. These benchmarks help customers make decisions on which vendor's server they will use to run their business on. There are benchmarks that attempt to measure the speed of personal computers. These types of benchmarks would be used by consumers to decide which vendor to purchase a computer system, laptop or tablet from. I have used the benchmarks described in this report to measure the speed of our Java SE Embedded products. Our customers run these benchmarks to

compare our performance to our competitors when deciding to select our products over an alternative.

14. Even minor performance gains justify decisions to invest months of engineering development to implement performance-delivering features. Since increased performance leads to increased sales, the engineering team at Oracle continues to invest heavily in R&D targeted at increasing performance. A large portion of our Virtual Machine engineering team is typically dedicated to performance improvement projects that span areas such as compilation techniques, garbage collection speed, startup improvement, synchronization improvements and performance measurement tools.

15. I will describe some industry accepted benchmarks on which such customer demand and engineering development decisions are based.

16. These are the same benchmarks used to test Android performance as discussed in my report:

CaffeineMark™ 3.0 * (http://www.benchmarkhq.ru/cm30)[1] PENDRAGON CAFFEINEMARK (tm) ver 3.0[2] (http://www.benchmarkhq.ru/cm30) Description: The CaffeineMark is a series of tests that measure the speed of Java programs running in various hardware and software configurations. CaffeineMark scores roughly correlate with the number of Java instructions executed per second, and do not depend significantly on the amount of memory in the system or on the speed of a computers disk drives or internet connection. For details on the CaffeineMark benchmark, refer to this URL http://www.benchmarkhq.ru/cm30/info.html. The CaffeineMark 3.0 scores presented in my report are in units of work completed in a fixed amount of time. This means that higher scores are superior.

[1] Please visit the CaffeineMark website cited to download PENDRAGON CAFFEINEMARK (tm) ver 3.0.

[2] The CaffeineMark benchmark was run without independent verification by Pendragon Software. Pendragon Software makes no representations or warranties as to the result of the test. CaffeineMark is a trademark of Pendragon Software.

compare our performance to our competitors when deciding to select our products over an alternative.

14. Even minor performance gains justify decisions to invest months of engineering development to implement performance-delivering features. Since increased performance leads to increased sales, the engineering team at Oracle continues to invest heavily in R&D targeted at increasing performance. A large portion of our Virtual Machine engineering team is typically dedicated to performance improvement projects that span areas such as compilation techniques, garbage collection speed, startup improvement, synchronization improvements and performance measurement tools.

15. I will describe some industry accepted benchmarks on which such customer demand and engineering development decisions are based.

16. These are the same benchmarks used to test Android performance as discussed in my report:

CaffeineMark™ 3.0 * (http://www.benchmarkhq.ru/cm30)[1] PENDRAGON CAFFEINEMARK (tm) ver 3.0[2] (http://www.benchmarkhq.ru/cm30) Description: The CaffeineMark is a series of tests that measure the speed of Java programs running in various hardware and software configurations. CaffeineMark scores roughly correlate with the number of Java instructions executed per second, and do not depend significantly on the amount of memory in the system or on the speed of a computers disk drives or internet connection. For details on the CaffeineMark benchmark, refer to this URL http://www.benchmarkhq.ru/cm30/info.html. The CaffeineMark 3.0 scores presented in my report are in units of work completed in a fixed amount of time. This means that higher scores are superior.

[1] Please visit the CaffeineMark website cited to download PENDRAGON CAFFEINEMARK (tm) ver 3.0.

[2] The CaffeineMark benchmark was run without independent verification by Pendragon Software. Pendragon Software makes no representations or warranties as to the result of the test. CaffeineMark is a trademark of Pendragon Software.

SciMark 2.0 (http://math.nist.gov/scimark2)[3] Description: SciMark 2.0 is a Java benchmark for scientific and numerical computing. It measures several computational kernels and reports a composite score in approximate Mflops (Millions of floating point operations per second). For details on the SciMark benchmark, refer to the URL http://math.nist.gov/scimark2. The SciMark 2.0 scores presented in my report are in units of work Mflops completed in a fixed amount of time. This means that higher scores are better.
kBench (Sun/Oracle Internal Benchmark) (attached as Exhibit A (Oracle Highly Confidential – Attorneys' Eyes Only) kBench (Sun/Oracle Internal Benchmark) Description: kBench is a collection of small Java benchmarks programs used to measure the relative performance of Java Virtual Machines. The collection of benchmarks included in the kBench distribution are sorting and hashing algorithms, string tests, (Sieve) prime number calculator, Queens, Hanoi, DeltaBlue and Richards benchmark tests. The kBench scores are in units of time (milliseconds). It measures the amount of time it takes to complete a fixed amount of work. This means that smaller scores are better. The data for the kBench comparisons are graphed and presented in my report, arranged in order to present the results from every benchmark program in a consistent way (where larger scores are better).

I had to select benchmarks that did not require any Java graphical APIs and could run on the APIs that are implemented in Android.

17. All the benchmarks were converted from standard jar files to dex jar files with, for example:

```
$ dx --dex --verbose --output=scimark-dex.jar scimark.jar
```

Each is attached as Exhibit B (CaffeineMark dex files), Exhibit C (SciMark dex files), and Exhibit D (kBench dex files) (Oracle Highly Confidential – Attorneys' Eyes Only).

18. This is because the Android Virtual Machine (Dalvik) executes dex files. This conversion does not change the performance benchmark suites or their analytical relevance.

B. Devices and Software Distributions

19. I selected two different hardware platforms to perform the tests on. Here are the specifications and links to the boards and operating system platforms that I installed and ran on these two devices:

[3] Please visit the SciMark website cited to download SciMark 2.0.

SciMark 2.0 (http://math.nist.gov/scimark2)[3] Description: SciMark 2.0 is a Java benchmark for scientific and numerical computing. It measures several computational kernels and reports a composite score in approximate Mflops (Millions of floating point operations per second). For details on the SciMark benchmark, refer to the URL http://math.nist.gov/scimark2. The SciMark 2.0 scores presented in my report are in units of work Mflops completed in a fixed amount of time. This means that higher scores are better.
kBench (Sun/Oracle Internal Benchmark) (attached as Exhibit A (Oracle Highly Confidential – Attorneys' Eyes Only) kBench (Sun/Oracle Internal Benchmark) Description: kBench is a collection of small Java benchmarks programs used to measure the relative performance of Java Virtual Machines. The collection of benchmarks included in the kBench distribution are sorting and hashing algorithms, string tests, (Sieve) prime number calculator, Queens, Hanoi, DeltaBlue and Richards benchmark tests. The kBench scores are in units of time (milliseconds). It measures the amount of time it takes to complete a fixed amount of work. This means that smaller scores are better. The data for the kBench comparisons are graphed and presented in my report, arranged in order to present the results from every benchmark program in a consistent way (where larger scores are better).

I had to select benchmarks that did not require any Java graphical APIs and could run on the APIs that are implemented in Android.

17. All the benchmarks were converted from standard jar files to dex jar files with, for example:

```
$ dx --dex --verbose --output=scimark-dex.jar scimark.jar
```

Each is attached as Exhibit B (CaffeineMark dex files), Exhibit C (SciMark dex files), and Exhibit D (kBench dex files) (Oracle Highly Confidential – Attorneys' Eyes Only).

18. This is because the Android Virtual Machine (Dalvik) executes dex files. This conversion does not change the performance benchmark suites or their analytical relevance.

B. Devices and Software Distributions

19. I selected two different hardware platforms to perform the tests on. Here are the specifications and links to the boards and operating system platforms that I installed and ran on these two devices:

[3] Please visit the SciMark website cited to download SciMark 2.0.

BEAGLEBOARD (Rev C4) Specifications
Texas Instruments OMAP3530 ARM® Cortex-A8™ Processor 720Mhz, 256MB RAM
Hardware specification URL:
http://beagleboard.org/hardware
Android 2.2 distribution for BeagleBoard was downloaded from here:
http://software-dl.ti.com/dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/02_00_00/index_FDS.html
The Ubuntu Linux Distribution for BeagleBoard was downloaded from here:
https://wiki.ubuntu.com/ARM/OMAPMaverickInstall

NVIDIA TEGRA-2 250 Developer Kit Specifications
Dual-core ARM® Cortex-A9 MPCore™ Processor, 1.0 GHz, 1024MB RAM
URL: http://developer.nvidia.com/tegra/tegra-devkit-features
The Android 2.2 and Linux Distributions for Tegra-2 can be downloaded from the Nvidia developer site at:
http://tegradeveloper.nvidia.com/tegra/downloads

20. The Beagleboard is a good test board because it is fully supported by both the Linux and Android community. TI is supporting an Android distribution and there is a supported Ubuntu distribution. This allowed me to run the same benchmarks on two different operating systems on the same hardware platform. This will also allow Google to validate these benchmark tests if they choose to.

21. The NVIDIA Tegra is a good test board because it is fully supported by NVIDIA. At the time of the benchmarking, they had both Android and Linux distributions for their device. This allowed me to run the same benchmarks on a second hardware platform under the two operating systems of interest. This will also allow Google to validate these benchmark tests if they choose to.

C. Android Sources

22. The Android sources that were used to build the Dalvik VM used during these tests were downloaded by following the Google instructions from this Google URL and specifying the froyo tag in the git command.

http://source.android.com/source/downloading.html
$ repo init -u git://android.git.kernel.org/platform/manifest.git -b froyo

23. The Android sources were built using the instructions located at this URL:

http://source.android.com/source/building.html

IV. PERFORMANCE ANALYSIS OF THE '104 PATENT ON ANDROID

A. Google Android benefits from RE38,104

26. I ran experiments to disable the Android functionality that Oracle accuses of infringing the '104 patent.

27. The baseline for these experiments was the Froyo release of Android, pulled from the Google git repository with the Google repo commands

```
$ repo init -u git://android.git.kernel.org/platform/manifest.git -b froyo
$ repo sync
```

to initialize and sync from the Google repository.

28. I then made two additional copies of the repository

(1) building side tables of resolved constant pool entries, but not quickening instructions

(2) not building side tables of resolved constant pool entries, nor quickening instructions

29. Modifications were made to the source in the copies to implement the experiments.

30. The modification for the first experiment (side tables but no quickening) are restricted to

dalvik/vm/analysis/DexOptimize.c

and consist of bracketing with #ifdef and #endif code that rewrites instructions to their QUICK forms. The modifications for the first experiment are shown in '104 Appendix 1.

31. The modifications for the second experiment (neither side tables nor quickening) included all the changes for the first experiment, plus additional changes to

dalvik/vm/analysis/DexOptimize.c

and some changes to

dalvik/vm/oo/Resolve.c

32. The changes for the second experiment also consist of bracketing with #ifdef and #endif code that builds the side tables of resolved constant pool entries. The modifications for the second experiment are shown in '104 Appendix 2.

33. Each workspace was compiled with

$ make -j2

34. The benchmarks used to test the performance of the workspaces were

(1) CaffeineMark™ 3.0 * (http://www.benchmarkhq.ru/cm30)

(2) SciMark 2.0 (http://math.nist.gov/scimark2)

(3) kBench (Sun/Oracle Internal Benchmark)

35. All the benchmarks were converted from standard jar files to dex jar files with, for example

$ dx --dex --verbose --output=scimark-dex.jar scimark.jar

36. I tested the performance of the repositories by running on the supplied Android emulator with

```
# dalvikvm -Xint:fast -cp /data/app/cm3-dex.jar CaffeineMarkEmbeddedApp
# dalvikvm -Xint:fast -cp /data/app/scimark-dex.jar scimark 2.0
# dalvikvm -Xint:fast -cp /data/app/kBench-dex.jar RunAll
```

I did not try running the trace compiler, because an examination of the source code indicated that it would not run without the side tables of resolved constant pool entries.

37. Each of the three workspaces ran all three of the benchmarks. I ran 10 iterations of each benchmark.

38. Separately, I built new workspaces with the modified source files, and ran them on a Beagleboard.

39. The accompanying charts attached as Exhibit E record the results of runs with the -Xint:fast command line option on the Beagleboard.

V. PERFORMANCE ANALYSIS OF THE '205 PATENT ON ANDROID

A. Introduction

44. This section describes the results of an analysis of Android's performance to measure the benefit of Android functionality that Oracle accuses of infringing the '205 patent to Android. The analysis consists of running Java-based benchmarks on Android platforms with accused features enabled and disabled in order to determine the impact that these features are providing Google.

45. There are at least two techniques in Android that are accused of infringing upon the '205 patent as I understand it. These are 1) Android's Just-In-Time compiler (JIT) and 2) Android's Inlining technique. The results included in the report show the benefit that each of these techniques provide and the methodology used to conduct the experiments.

46. In addition to the Android measurements, I also measured the benefit that Oracle realizes in its use of these patented techniques by executing the same set of benchmarks on its Java SE Embedded product running on the same selected hardware devices.

B. Methodology

47. The methodology used to produce the results consisted of the following steps:

(1) Identify hardware devices capable of running Android and a standard Linux distribution.

(2) Download prebuild Android and Linux distributions.

(3) Download Android Froyo 2.2 sources.

(4) Identify Java benchmarks that are capable of executing on both Android and Linux.

(5) Run benchmarks on Android with Just-in-time compilation (JIT) enabled and disabled

(6) Run benchmarks on Android with Inlining of Native methods enabled and disabled

(7) Run benchmarks on Linux with JIT enabled and disabled

(8) Run benchmarks on Linux with Inlining enabled and disabled

(9) Gather results and produce charts

All of the collected data for the full set of benchmarks are available in the spreadsheet attached as Exhibit F.

C. JIT Performance Testing

48. To test the benefit of the Android JIT, I used a dalvikvm command line option to enable and disable the JIT. The dalvikvm VM supports an option –Xint:{portable,fast,jit} which causes the dalvikvm to run Java bytes codes using one of three methods.

–Xint:portable	Portable "C" based interpreter with no JIT
-Xint:fast	Assembler based faster interpreter with no JIT
-Xint:jit	JIT enabled with the assembler interpreter

49. To test the performance of the JIT, I ran the three selected benchmarks using the "–Xint:fast" mode and the "–Xint:jit " mode. These tests show the performance difference that the JIT provides above and beyond interpreter only.

50. Again, before the benchmarks could be run on the device, the benchmarks needed to be converted to dex format, since this is the only format that Android supports. Here is the command I used to perform this operation. The "dx" command is available from the Android SDK.

```
% dx --dex --output=benchmark-dex.jar benchmark.jar
```

51. To copy the converted benchmark to the Android device, I used the adb tool, also available from the Android SDK.

```
% adb push benchmark-dex.jar /sdcard/benchmark-dex.jar
% adb push run-benchmark.sh /sdcard/run-benchmark.sh
```

52. To run the benchmark on Android, I used a shell script (run-benchmark.sh) that I wrote which would allow for 10 runs of each test.

53. Before starting each benchmark run, the script cleans out the dalvik-cache, syncs the storage media and then runs each test 10 times. I executed this script using this set of commands:

```
% adb shell
$ mkdir /dev/tmp
$ sh /sdcard/run-benchmark.sh >/dev/tmp/output.log
```

The content of the run-benchmark.sh script is as follows:

```
"Ljava/lang/Math;", "cos", "(D)D" },
{ javaLangMath_sin,
"Ljava/lang/Math;", "sin", "(D)D" },

#endif
};
```

56. Once I created the two dalvikvm binaries, I ran the benchmarks the same way as in the JIT runs except I replaced the original dalvikvm (libdvm.so) with the unmodified version followed by the modified VM prior to performing each test run. All of the collected data for the full set of benchmarks are available in the spreadsheet attached as Exhibit F.

E. Device Specific Notes

57. In order to ensure that there was little to no CPU activity while the benchmarks were being run, I commented out the zygote process from init.rc to keep the Android stack from launching. This still allowed the adbd server to run which enabled me to connect to the device via adb.